2022 XVLIII Latin American Computer Conference (CLEI) | 978-1-6654-7671-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/CLEI56649.2022.9959901

# Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue

Raimundo N. V. Diniz-Junior*, Caio César L. Figueiredo†, Gilson de S. Russo ‡, Marcos Roberto G. Bahiense-Junior§, Mateus V. L. Arbex ¶, Lanier M. dos Santos‖, Raimundo F. da Rocha**, Renan R. Bezerra††, and Felipe T. Giuntini$^{x}$
Sidia R&D Institute
Manaus, Brazil

Email: *{raimundo.junior, †caio.figueiredo, ‡gilson.russo, §marcos.bahiense, ¶mateus.arbex, ‖lanier.santos, **raimundo.rocha, ††renan.bezerra, ‡‡felipe.giuntini} @sidia.com.

*Abstract*—**The expansion of online services and the increasing demand for a good experience on the Web have promoted the emergence of different rendering frameworks based on JavaScript that assist the agile development and optimize the application's performance. This paper presents a methodology to evaluate web rendering frameworks based on virtual and incremental DOM paradigms: Angular, React, and Vue. We conducted a study case based on a generic toy application and analyzed the build size, time to interact, and the DOM manipulation time aspects. Our results show that Vue is 758% faster than React and 595% compared with Angular in manipulation time. Angular occupies 54% more bundle space than Vue and 45% than React. React shows the best time of interactive (300ms), 33% faster than Vue and 50% than Angular. The experiments can support future Web project methodology decisions.**

*Index Terms*—**Web rendering frameworks, incremental and virtual DOM, front-end, React, Vue, Angular; benchmark, performance avalution, web development, JavaScript**

## I. Introduction

The growth of the web in the last years allowed observing the advancement of models and frameworks to contribute to different fronts and features of web applications development. For example, agile and process improvements methodologies [1]–[3], global software development challenges [4]–[6], quality and security aspects [7]–[10].

Regarding the general support of web development, different rendering frameworks based on JavaScript have been proposed to improve the creation of Single Page Applications-SPAs and facilitate the development of easy-to-use interfaces. Frameworks like Vue [11], Angular [12] and React [13] provides the means to support the process of developing and maintaining multi-platform web apps by enabling multiple APIs for the developers.

The comparison between popular frontend rendering frameworks is often discussed, especially performance and the learning curve. These tools are based on different architectures and implement rendering techniques, ie. Virtual DOM or Incremental DOM. Besides, constant updates can overhaul certain aspects of the whole toolset. Meanwhile, the evolving nature of the web also means that obsolescence is a prescient factor, and the need to follow current trends or update legacy features also impacts the choice of the equivalent framework to develop [14], [15].

When we analyze how the frameworks render the HTML, we have two schools of thought: incremental DOM and virtual DOM. The virtual DOM (VDOM) is a programming concept where an ideal representation, or 'virtual', of the node tree is kept in memory and synchronized with the 'real' DOM by a library, like the ReactDOM. This process is called reconciliation, which is the foundation of the VDOM [13]. When We instantiate Vue, the node tree is called VNODE, working on a custom logic to prevent unnecessary re-renders [11]. In contrast, in the incremental DOM, which is the technique that Angular uses, each separate component is compiled in order and, after the node tree is mounted on the real DOM, interpreted directly by the browser. Angular also provides an additional syntax to apply dynamic values to its components and update the DOM freely, rendering the new state as the values are applied [12].

React's implementation of virtual DOM aims to solve one of the main problems: the performance hit when rendering HTML components. Since VDOM keeps a virtual node tree on memory, it can differentiate when a component should re-render based on the actual state of the application, impeding constant and unnecessary DOM updates. This technique is efficient for smaller applications due to the memory cost associated with keeping a virtual node tree readily available on memory can generate a performance debt on complex apps [16].

Similar to React, Vue framework uses the same rendering technique, based on Virtual DOM, but with an MVVM pattern architecture (*Model, View, View-Model*). A difference to be considered compared to React is how data binding works on

This work was partially supported by Samsung Eletrônica da Amazônia Ltd., under the stimulus of the Brazilian Informatics Law No. 8.387/91.

Vue, React supports one-way data binding, passing properties from the parent to the child component, while Vue and Angular allows two-way data-biding, which makes handling application states easier to developers. [11]–[13].

The Angular framework supports a more robust structure, focused on the MVC pattern *Model, View, Controller*, in addition to the fact that by default, the latest versions come with the Typescript boilerplate [12], [14]. The main difference between Angular and the previous frameworks is how Angular renders its components. Based on incremental DOM, Angular provides a syntax to bind the components to the compilation step (decorators) that creates or updates all components one by one. In this scenario, the real DOM detects new node trees, compiles each component in a set of instructions, observes any alterations, and then applies it. The idea behind not generating a copy of the tree to compare gives an advantage when analyzing the memory consumption. However, since it needs to compile sequentially, developing a set of instructions for every new or old node can impact the speed of any DOM manipulation [14].

To the best of our knowledge and from what is known in the scientific literature, the works have focused on (i) descriptive comparison of framework characteristics and (ii) performance tests in particular contexts with little experimentation data, without considering aspects of the iteration time and the bundle size. In addition, there are few scientific works, mostly gray literature. However, evaluating different frameworks in a generic application can yield more reliable results.

Considering this problem and the lack of good experiments in the literature, this paper aims to evaluate the performance of web rendering technologies based on JavaScript, considering the time to interact, the DOM manipulation time, and the bundle size. We look for the principal evaluation metrics in the literature and analyze virtual and incremental DOM by creating a standardized application. We test DOM manipulations using three frameworks representing these two rendering techniques: Angular for incremental and React and Vue for virtual. We highlight the main contributions:

- A methodology to compare incremental and virtual Web rendering frameworks based on JavaScript.
- A comparative case study with Angular, Vue, and React frameworks.
- A generic and toy example Web application to support benchmark analysis.

**Paper Outline:** This paper is organized as follows: The section II discusses the related works that describe and evaluate the performance of rendering frameworks based on JavaScript language. Section III presents the methodology with the evaluation metrics, environment configuration details, and the case study. The section IV presents and discusses the results and comparisons between the web technologies. The section V discuss the takeouts, limitations, future works, and conclusions.

## II. Related Works

This section presents the related studies about the performance comparison of rendering web technologies.

The work [17] performs a comparative analysis between the AngularJS and React JavaScript frameworks in order to define the best choice for web application development. The authors define AngularJs as a framework capable of creating web applications based on HTML, CSS, and JavaScript using modularization and dependency injection resources. On the other hand, React is defined as a library that facilitates the creation of interfaces using reusable components. It was pointed out attributes such as DOM, Learning Curve, packaging, Binding, and Templating, among others, to compare the frameworks. The authors conclude that AngularJS presents a complete structure with resources not present in React; however, with the use of VirtualDOM, React has a better performance.

The work [18] makes a comparative analysis between Angular, Ember, Knockout, and Backbone. Such a study compares these technologies in terms of internet popularity, bundle size, and performance. The most representative framework on the internet was Angular, reaching the highest number of searches on the Google platform, with approximately 105 million results, on Stack Overflow with approximately 252,317 queries, on YouTube with about 800,000 views 36,669 stars on GitHub. On the other hand, the minor representation of internet searches is split into two frameworks. Regarding Google searches, Ember shows up with approximately 6 million results, considering queries on Stack Overflow, stars on GitHub, and views on YouTube Knockout with 19,316, 5,693, and 17,000 results, respectively. Another metric used in work is the framework's build size. In this requirement, Backbone has generated smaller builds with 6.5KB in contrast to Ember, which creates builds of 90KB. The Angular bundle reached 39.5KB and the Knockout with 55KB. Furthermore, a simple structure was proposed regarding performance tests, printing 500 numbers and comparing the number of operations per second on all four frameworks. The results obtained show Angular reaching 8,732 operations per second, becoming the fastest one, followed by Backbone with 75.06, Ember with 5.56, and finally Knockout with 1.18 operations per second. Lastly, the frameworks explored on [18] paper are based on MCV architecture, which provides a readable code, which is very important when developing complex applications. [18] reports negative points of each studied framework related to safety, performance, and documentation.

In [19] the authors conducted a survey about developing an application which may support undergraduates students, assigned to a web development course, comparing modern structures and technologies for front-end development, aiming to find a proper library or framework to be used for coding. The authors analyze the React, Angular and Vue frameworks, considering the most download rate technologies according to the 2018 StackOverflow survey, (Angular 36,9%, React 27,8%). Besides, the criteria is the component based structure, programming language, modeling, state management approach, data binding, multi-platform support and extensive documentation, have been considered.

The angular stood out from other frameworks, mainly due to the development platform that incorporates the necessary elements to allow the development of applications with a high level of complexity, in addition to simultaneously allowing its scalability, a technical documentation and business support.

The authors do not conduct a performance evaluation about the frameworks, they just compare the public informations and the documentations about the frameworks.

The work [14] makes a comparison between the JavaScript Angular and Vue.js frameworks in order to find the advantages and disadvantages of using each framework. For this, two To-Do list applications were developed, one with Vue. js and another with Angular in order to perform the comparisons. A To-do list is an application where the user can register their activities and mark those that have already been completed. The metrics used for the comparison were: loading, rendering, painting, scripting, system, and idle. The results show that Vue.js has more performance than Angular, keeping its general average of results between 0 and 1000ms, while Angular was above 2000ms. Based on the results [14] concludes that Vue.js is a recommended framework for beginner programmers and small projects, and Angular is ideal for large scalable projects.

In the work [15], the authors compare Vue and React in terms of tools complexity, popularity and practice. Since, both implements the Virtual DOM concept, the benchmark considers the performance and architecture of each one.

The test has been conducted implementing a Calculator app on both frameworks and evaluating its response time, thus Google Chrome Performance tool has been used in order to collect data. During the tests, there was a slight advantage of React which presented Scripting time of 279ms and Rendering time of 6ms while VueJs shows up Scripting time of 336ms and Rendering time of 9ms.

However, it's needed to highlight that, such a tiny difference between these frameworks performance is not enough to be considered, since results may diverge, depending on tests structure [15].

In the work [20], the authors conduct the Benchmark using the Lighthouse  [21], a Google open-source tool that tests and evaluates the quality of web apps in terms of performance, accessibility, good practices, and SEO *Search Engine Optmization*. They conduct the tests in two 'ToDo' Single Page Applications based on Angular and Svelt frameworks and in mobile and desktop environments, using an aggregate score provided by the Lighthouse tool. The results show that Svelte received 98 as the score in a mobile environment while Angular scored 90 points. Svelte received the maximum score in a desktop environment while Angular scored 90 points, all conducting the same tests. Therefore, based on Lighthouse's results, [20] point out that the Svelt advantage over Angular, regarding the mentioned parameters. Compared to this current paper, it is possible to highlight the number of metrics used to define the research results and the use of the Lighthouse tool. Also, a benchmark was performed for testing at the code level, considering processing time and respecting the life cycle of each framework.

In the work [22], the authors point out three main concerns regards the challenges of comparing JavaScript frameworks, (i) several choices, (ii) several execution environments, and (iii) outdated information. To solve this problem, the authors present the JACT, a tool that provides a source code comparison feature and facilitates individual understanding of characteristics related to each framework. JACT tests performance measurement at runtime and allows sending tasks to the source code that uses specific JavaScript Frameworks. JACT compares aspects related to each framework as (i) task-based comparison, *i.e.,* the actor can make a comparison based on simple front-end development tasks such as DOM manipulation and AJAX [23]; (ii) source code comparison, i.e. the actor can compare the source code for the task's implementation; (iii) runtime performance measurement for each selected task source code, *i.e.,* can measure the processing time to finish the task 100 times; (iii) task submission, *i.e.,* the are the option to send the tasks where an author can specify which tasks to send by filling a form; and (iv) *i.e.,* source code submission, *i.e.,* the framework can adapt itself for newer versions. The experiment consists of changing the text on a page using three JavaScript frameworks: VanillaJs (Pure JavaScript), jQuery (v3.3,v3.4), and Vue.js (v2.5,v2.6), the browsers Google Chrome and Firefox, and the iOs, Android, Windows, and Mac OS system devices. The experiment was performed 5 times, after that, the mean was calculated.

The results compared to jQuery show a minimal difference from vanilla. Regarding the browsers, the performance of firefox tends to be higher compared to chrome. The results shows that Vue.js 2.5 presented the worst performance, up to 40 times slower than VanillaJs, on Windows PC and Chrome environments.

The author [24] show us a perspective related to the use of JavaScript frameworks for the development of mobile applications, comparing hybrid and mobile only apps. Considering the problem of choosing the best framework for a specific objective, the paper shows the optimal choices in frameworks for native development.

The following criteria was selected for the analysis: Learning curve, documentation, development cost, emulators and debugging, response time and speed, market value, code reuse and teamwork, maintenance and updates. The author did simple tests as the installation and creation for small components for the analysis, basing on the criteria mentioned previously, where the results were presented on the Likert's scale, varying from 1 to 5. Analyzing the results, the authors conclude that hybrid solutions are closer to native in relation to the runtime speed requirements and content rendering. The paper [24] highlights hybrid solution as the best choice for MVP *Minimum Viable Product*, as it is a simple version with minimal development.

In our paper, we bring some comparisons applying similar metrics in addition of other items such as TTI and bundle size. Some of the related works cited, such as [17] and [19], don't use a standardized application to compare the frameworks, these papers just discuss the main descriptions related with each framework.

For a better understanding of these Frameworks, a table has been provided TABLE I, showing applications, created by their respective authors, for evaluating topics such as life cycle, scripting, rendering, description and system.

## III. Methodology

In this section, we present our methodology to benchmark the frameworks mentioned previously. We conduct a performance evaluation consisting of functions: create, edit, and delete certain quantity of elements on the browser's window, following the lifecycle (*The framework's function controlling the states of an application*) proposed by each framework and library.

### A. Evaluation metrics

We apply the following evaluation metrics to conduct the performance analysis of Vue, Angular, and React Web rendering frameworks. The metrics were chosen considering the development of applications and the use perspective.

1) **Build bundle size:** The application bundle is the web-optimized production build created from each framework's build tools. Their toolsets result in an HTML file and a web-packed minified JavaScript file, ready for production deployment. For the native JavaScript, we utilized a static HTML file that encapsulates the equivalent script, containing the functions we wrote for the frameworks. The bundle was collected by creating the production build for each framework and then checking the size by analyzing the file's properties. The smaller the file, the faster the download and the fewer elements for the browser to analyze. [12] [13] [11] .
2) **Time to Interactive:** The Time to Interactive (TtI) is defined as the period between when the last long task (task longer than 50ms) finishes and the quiet window starts (usually a period of inactivity of at least 5 seconds) [25]. The TtI is a metric to give an overview of when is ready to accept interaction after DOM manipulation from the user's perspective as the main thread is not blocked. The lower the tti, the faster the application is available for user interaction. To evaluate, we utilize the Lighthouse tool, which automate and returns a collection of metrics to score a web page based on a rank.

$$LTT - QWT = TtI$$

LTT is the time after the final long task finishes and the QWT is the time when a 5 second of idle window starts. The interval between them is the Time to Interactive.
3) **DOM Manipulation time:** Each framework has its native lifecycle hook, separating the process of creating, updating, and destroying a component into distinct methods while providing the means to implement custom logic for manipulating the DOM tree. A faster lifecycle also means a shorter rendering time. We considered the time measurement between the initial state and when the framework calls its equivalent hook after the component updates. On React, we have both methods `componentWillMount()`, and `componentDidUpdate()`, called before the component binds to the DOM and after the states changes, respectively. Inside Angular, we have `ngAfterContentInit()`, which fires after the child content is loaded, and `ngAfterViewChecked()`, which fires after the change detection checks the template for changes. Contents refer to external components projected on the current component, while view refers to the template rendering. For Vue, we have `beforeCreate()`, called when the instance finishes the properties resolution but before processing the bound data, and the `onUpdated()` signals when DOM processing ends. This metric demonstrates the most effective framework for manipulating the DOM.

### B. Configuration environment and Technologies

We conduct the performance evaluation experiments with Angular, React, and Vue frameworks. For the benchmark environment, we utilized a desktop with Intel Xeon E-224G 3.50GHZ CPU, 32GB RAM, and Ubuntu 20.04.3LTS. Besides, we use the following tools: Docker v20.10.12 [26], NodeJS 14.18.2 [27], Nginx 1.21.6 [28], and Selenium-web driver 4.1.1 [29]. An application was developed for each framework to validate the case study and collect data as the foundation for testing.

We implement the tests inside the docker container using the production build of each framework. The Nginx tool allows a proxy instance to serve the web applications. The docker image for the testing environment contained only the OS System and its initial configurations, the node installation, and the Node Packet Manager (NPM). All tests occurred on the same machine using the same procedure.

### C. Evaluation Procedures

Both React and Vue use virtual DOM as their essential rendering technique. Alternatively, Angular uses the incremental DOM technique. Therefore, we defined a standard structure composed of basic HTML tags. We defined four main DOM manipulation functions: create, edit one, edit all, and delete all. These functions are bound to the button nodes to create a list of HTML elements to attach as a child node on the HTML body, edit a component inside the created list, modify all existing elements inside the list, and remove all HTML nodes of the list.

All the mentioned frameworks have an initial reactive state for when the component is available after being attached to DOM mounted. This state is observed by the lifecycle and reacts to every change. We store the execution time when the DOM manipulation method is fired at the lifecycle hooks and after the state change function is called. The difference between those times is logged.

TABLE I
SUMMARY OF RELATED WORKS

| Study | Framework | DOM Type | ITT | Bundle | Description | Performance | Application Context |
|---|---|---|---|---|---|---|---|
| [14] | Angular / Vue | Incremental/Virtual | | | ✓ | ✓ | Web - To Do List APP |
| [15] | React / Vue | Virtual | | | ✓ | ✓ | Web - Calculator APP |
| [17] | Angular / React | Incremental/Virtual | | | ✓ | | Web - No Aplication |
| [18] | Angular / Ember / Knockout / Backbone | Incremental | | ✓ | ✓ | | Web - No Aplication |
| [19] | React / Angular / Vue | Incremental/Virtual | | | ✓ | | Web - Blueprint of educational |
| [20] | Angular / Svelte | Incremental | | | ✓ | ✓ | Web - TODO APP |
| [22] | JavaScript / jQuery / Vue.js | Incremental | | | ✓ | ✓ | Web - Jact |
| **Our Evaluation** | Angular / Vue / React / Vanilla | Incremental/Virtual | ✓ | ✓ | ✓ | ✓ | Web - Render APP List |

For testing purposes, a Selenium [30], [31] script was developed for automation purposes to make 36 individual attempts for each DOM manipulation function described before and using three different sample sizes: 1000, 10000, and 50000 HTML elements. Each attempt was made inside our Docker container, using an nginx image to set up the environment, containing the production build for each framework. After each attempt, the selenium writes the logs to a file, in which we extract the average time individually.

For TtI, we applied the Lighthouse tool [21] provided by the Chrome browser, in which we continuously tested first-page load on the default state of the application for each separated framework. We extracted the resulting JavaScript artifacts from each framework build production tool for the build size bundle.

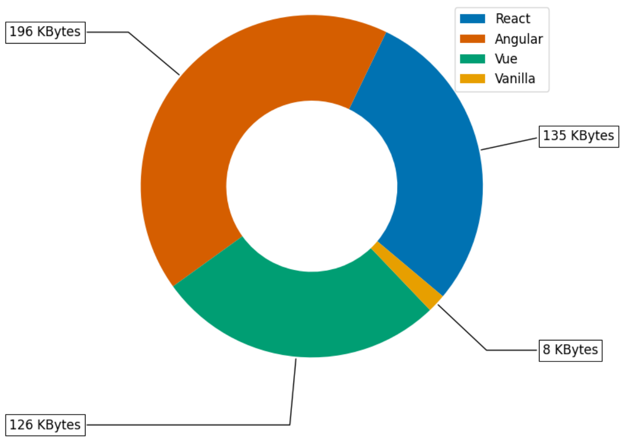

Fig. 1. Pie chart the represents the build size bundle for each framework and the HTML file for the vanilla JavaScript.

## IV. Results

In this section, we describe the results based on the methods detailed on section III.

### A. Build bundle size

The bundle size generates after the build is essential for a lightweight application. As noted in figure 1, the framework with the most extensive bundle is Angular with 196.608 bytes, followed by React with 135.168 bytes and Vue with 126.976 bytes. In this context, the Vue library stands out as the lightest build size, meaning less time to download the bundle when requested by the page. Angular has the highest bundle size due to its innate structure that follows a more standard MVC pattern, while React and Vue are focused on being freeform and lightweight [11]–[13].

### B. Time to interactive

As shown in chart 2, the results for TtI were close, spanning the frameworks. Since the application does not have any dynamic loading or API call, the first-page load is not as extensive as some Web applications and provides a natural fast TtI. As for the highlights, React presents the fastest result, having the lowest time of 0.3s, coming as fast and lower than the native JavaScript. In contrast, Angular has the highest time compared to the other results, becoming ready for user interaction in 0.6s. Vue does present a similar result to native JavaScript, having the same approximated time of 0.4s.

### C. DOM manipulation time

The chart 3 represents the results of the test attempts when manipulating 1.000 simultaneous HTML elements. With this scenario in mind, we can highlight Vue as the framework with the lowest creating time compared to the other frameworks, losing only to vanilla JavaScript. On to DOM updates, the incremental DOM shows a better result, having the lowest time in editing one element and editing all, compared to the virtual DOM frameworks. React, and Vue have lower times than Angular on deletion, which sits above 4 ms for deleting all the elements created.

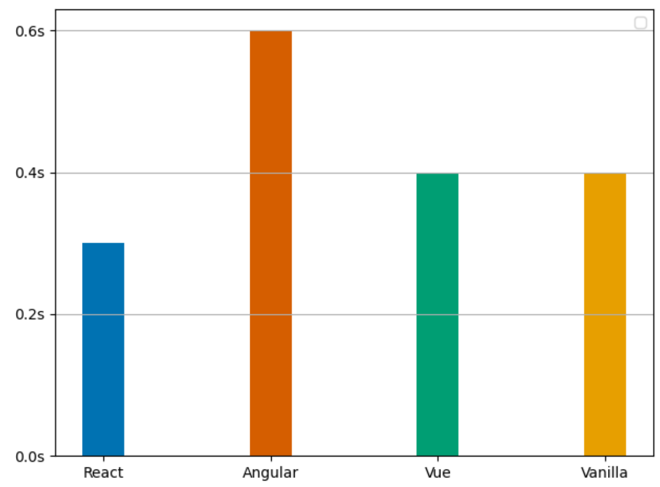

Fig. 2. Bar chart that represents the TtI results between the frameworks and the native JavaScript.

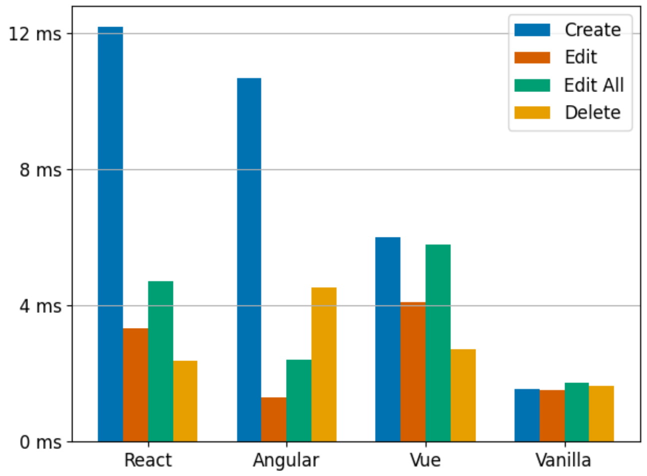

Fig. 3. Test attempts with 1.000 rendered elements.

With 10.000 simultaneous elements, represented in figure 4, it is possible to highlight two recurring cases through the tests. The first happens when attempting the create function in the React application. It stands out as the highest value among the others, measuring at 254.10ms on average. Although the highest time happens with React, the lowest also happens with a virtual DOM framework, with 36.5ms on average for Vue. The other recurring case is with deleting all elements on Angular, given its exceptionally high time. Angular, on average, takes 195.2ms to complete the removal cycle of all elements. It is important to note that Angular also has better performance in updating DOM elements. For the vanilla JavaScript, the times average is the same across all functions.

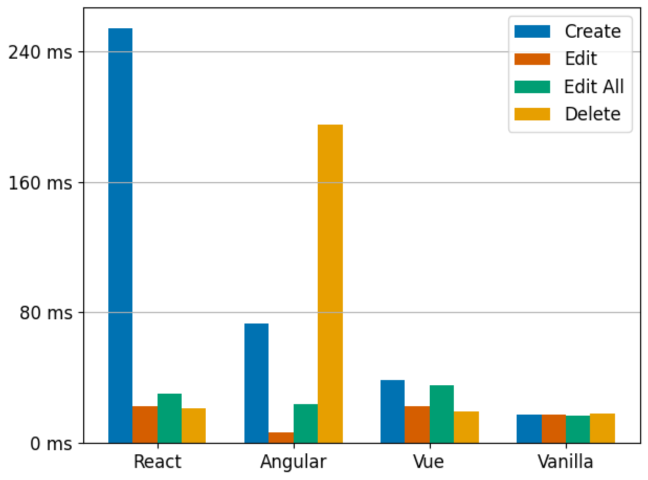

Fig. 4. Test attempts with 10.000 rendered elements.

Besides, the test attempts with 50.000 elements (figure 5) are the most taxing since the element load severely impacts the performance overall. Vue shows a consistent time concerning its previous attempts, having the lowest creation and deletion time while maintaining a solid editing time. Again, React takes the longest to create the 50.000 elements with 5165ms, and Angular takes 4043ms to delete all while maintaining the lowest update time. Comparing React and Angular shows they both have strengths and weaknesses on opposite functions, while Vue maintains a solid time across all functions, similar to native JavaScript.

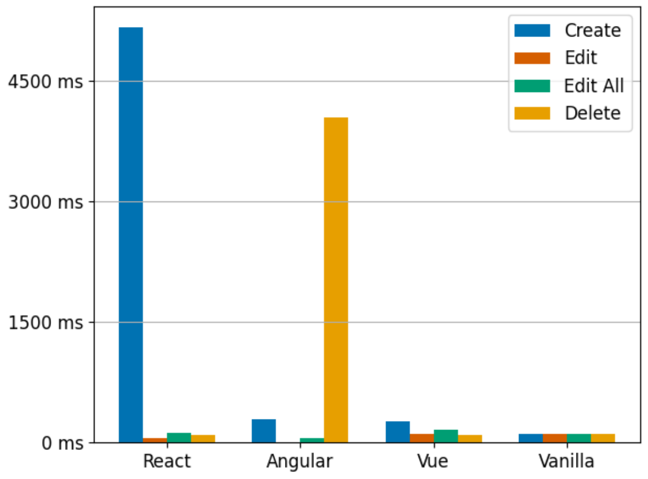

Fig. 5. Test attempts with 50.000 rendered elements.

The overview in figure 6 shows us the aggregation of all test results. As mentioned previously, times are similar in most cases, save for two exceptions. Mostly, results scaled as expected, and it was apparent that increasing the element count had different impacts depending on the framework.

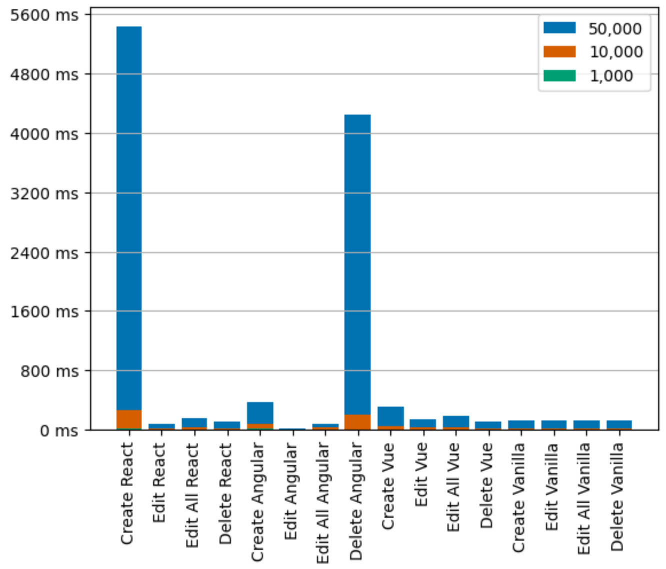

Fig. 6. Overview of the test attempts.

It is worth noting that since vanilla is pure JavaScript, it is expected to show superior performance in every attempt. On the other hand, developing using only pure JavaScript may be costly since the framework's primary purpose is to streamline the process of creating an application. On the table II we can observe the results considering the response time of adopted frameworks. Vue and react, due to usage of Virtual DOM strategy, presented the best execution time, considering obtained results on creation and deletion scenarios, Vue performed 90% and React 20% faster than Angular, regarding creation and deletion methods.

TABLE II
TIME PERFORMANCE OF CREATE AND DELETE FUNCTION

| | **1000** | **10000** | **50000** | **Total** |
|---|---|---|---|---|
| **Angular** | 14,3 | 261,9 | 4337,9 | 5544,9 |
| **React** | 15,3 | 275,05 | 5255,6 | 4615,1 |
| **Vue** | 8,35 | 55,9 | 364,9 | 429,1 |
| **Vanilla** | 3,2 | 35,1 | 205,4 | 243,7 |

In terms of editing functions, when there's a rendered element on the screen, as Angular applies incremental DOM edition strategy, it can manipulate the existing elements in a more performative way, such as Angular presented itself 60% faster, related with React and 71% related with Vue, as exhibited on III.

TABLE III
TIME PERFORMANCE OF EDIT FUNCTION

| | **1000** | **10000** | **50000** | **Total** |
|---|---|---|---|---|
| **Angular** | 3,6 | 29,8 | 61,0 | 94,4 |
| **React** | 7,8 | 51,7 | 179,4 | 239,0 |
| **Vue** | 9,7 | 55,6 | 267,9 | 333,3 |
| **Vanilla** | 3,2 | 34,7 | 203,8 | 241,7 |

Finally, the results provided by this section pointed out that regarding the performance of the frameworks, in terms of DOM manipulation time at the highest element count, Vue was 758% faster than React and 595% compared with Angular. Concerning the build size it is possible verify that Angular occupies a larger bundle space by 54% compared to Vue and 45% with React. In relation to time to interactive the React shows a time of 300ms, faster than Vue and Angular by 33% and 50% respectively.

So, based on the results, we recommend Vue for general use in applications that demands heavy DOM manipulation, especially when removing and creating elements. For Angular, we recommend using for applications that needs to constantly update DOM elements. For React, we recommend for applications that need lighter and interactive environments.

## V. Conclusions

In this paper, we presented a performance evaluation between distinct frameworks for rendering HTML elements on the DOM, one of them being the virtual DOM and the incremental DOM, implemented by frameworks like React, Vue, and Angular. Data such as HTML elements rendering time, time to interact, and build bundle size were collected. An application web was developed for each mentioned framework and a native JavaScript as a base parameter for comparison. With equivalent implementations and respecting the rules of the lifecycles, determining the end of operations, we extracted the metrics detailed in section III from the developed web applications.

The results show that React had difficulties creating the elements to the DOM since Angular and Vue average around 270-290ms while React is above 5000ms. Both are based on virtual DOM and had different results. Nonetheless, we can see that Angular, with its implementation of incremental DOM, showed us similar results when creating elements compared to Vue frameworks, averaging a difference of about 20ms on 50,000 elements and in editing values inside the DOM scenarios, Angular had the best performance. In light of that, it is possible to see the main advantage against VDOM or VNodes: To update nodes without the need of an intermediary to match the elements in memory.

In assessing the data extracted, we conclude that the solution proposed by the virtual DOM implemented inside the Vue framework had a slight advantage overall, having a good performance in every function and maintaining peaks under the 1s mark in the largest sample size.

It is worth mentioning that our test scenario is based on the standard version of the selected frameworks without the use of third-party libraries or modules that can assist in the rendering process when dealing with a massive load of HTML elements, such as clustering the elements together, rendering only visible elements (Virtual lists), or pagination. These methods were not used to provide a benchmark environment where we have a constant and stressful rendering scenario. All frameworks involved have their characteristics inside their core rendering, control layers, structure, and lifecycles, which may impact the HTML rendering directly.

The results (section IV) give a perspective on the behavior of the solutions in our test scenario.

We highlight the main contributions:

- We develop a methodology to compare different Web render frameworks based on JavaScript ( incremental and virtual), observing the main metrics in the literature.
- We provide a comparative case study with Angular, Vue, and React frameworks using a non-vies application, supporting the project's decision of technology choice by companies and developers.
- We provide a standalone Web application to support new benchmark analysis.

As future works we envision:

- Develop new metrics based on formal computing studies to support the Web performance test area.
- Explore performance testing in large-scale project environments.
- Develop intelligent mechanisms that data about software requirements support technology choice.
- Check mechanisms to provide optimizations in pre-existing software.
- Study of JavaScript frameworks directed to the development of the mobile applications.

## References

[1] A. F. Cruz, C. P. Godoy, L. M. Santos, L. F. Marinho, M. S. Jardim, E. P. Silva, C. A. L. Pahins, P. Fonseca, and F. T. Giuntini, "Blueprint model: An agile-oriented methodology for tackling global software development challenges," *Advances in Science, Technology and Engineering Systems Journal*, vol. 5, pp. 353–362, 2020.

[2] Y. Zheng, Y. Liu, X. Xie, Y. Liu, L. Ma, J. Hao, and Y. Liu, "Automatic web testing using curiosity-driven reinforcement learning," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 423–435.

[3] S. C. Fonseca, M. C. Lucena, T. M. Reis, P. F. Cabral, W. A. Silva, F. de S. Santos, F. T. Giuntini, and J. Sales, "Automatically deciding on the integration of commits based on their descriptions," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 1131–1135.

[4] W. Moreno, P. Afonso, and H. Costa, "Challenges and solutions of project management in distributed software development," in *2019 XLV Latin American Computing Conference (CLEI)*, 2019, pp. 1–10.

[5] F. Santos, J. Sales, F. Giuntini, and C. Godoy, "A preliminary analysis of communication preferences in a global software development environment," in *The 39th ACM International Conference on Design of Communication*, ser. SIGDOC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 357–358. [Online]. Available: https://doi.org/10.1145/3472714.3473908

[6] A. O. Penha-Junior, C. E. De Souza, G. M. Souza, J. S. Araujo, J. I. B. Vilarouca-Filho, L. M. Barroso, R. j. B. Fernandes, P. C. R. Fonseca, and F. T. Giuntini, "Challenges in the development of a global software user interface by multicultural teams: An industrial experience," in *2021 2nd Asia Service Sciences and Software Engineering Conference*, ser. ASSE '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 41–47. [Online]. Available: https://doi.org/10.1145/3456126.3456144

[7] M. Bruno, P. Ibañez, T. Techera, D. Calegari, and G. Betarte, "Exploring the application of process mining techniques to improve web application security," in *2021 XLVII Latin American Computing Conference (CLEI)*, 2021, pp. 1–10.

[8] P. Matsubara, I. Steinmacher, J. Maldonado, B. Gadelha, and T. Conte, *Trust Yourself! Or Maybe Not: Factors Related to Overconfidence and Uncertainty Assessments of Software Effort Estimates*. New York, NY, USA: Association for Computing Machinery, 2021, p. 452–461. [Online]. Available: https://doi.org/10.1145/3474624.3474643

[9] D. A. da Silva and W. M. Watanabe, "Cross-browser incompatibilities classification layout: A comparative study between different models," in *2021 XLVII Latin American Computing Conference (CLEI)*, 2021, pp. 1–10.

[10] S. a. Santos, B. Silveira, V. Durelli, R. Durelli, S. Souza, and M. Delamaro, *On Using Decision Tree Coverage Criteria For Testing Machine Learning Models*. New York, NY, USA: Association for Computing Machinery, 2021, p. 1–9. [Online]. Available: https://doi.org/10.1145/3482909.3482911

[11] Vuejs.Org. (2022) Nodes, trees, and the virtual dom. Acessed in 02 feb. 2022. [Online]. Available: https:vuejs.org/v2/guide/render-function.html

[12] M. License, "Angular applications: The essentials," 2022, acessed in 02 feb. 2022. [Online]. Available: https:ioguidewhat-is-angular

[13] R. Org.

[14] O. C. Novac, D. E. Madar, C. M. Novac, G. Bujdosó, M. Oproescu, and T. Gal, "Comparative study of some applications made in the angular and vue.js frameworks," in *2021 16th International Conference on Engineering of Modern Electric Systems - EMES*, 2021, pp. 1–4.

[15] C. M. Novac, O. C. Novac, R. M. Sferle, M. I. Gordan, G. BUJDOSó, and C. M. Dindelegan, "Comparative study of some applications made in the vue.js and react.js frameworks," in *2021 16th International Conference on Engineering of Modern Electric Systems (EMES)*, 2021, pp. 1–4.

[16] "Performance optimization techniques for reactjs," in *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 2019, pp. 1–5.

[17] A. Kumar and R. K. Singh, "Comparative analysis of angularjs and reactjs," *International Journal of Latest Trends in Engineering and Technology*, vol. 7, no. 4, pp. 225–227, 2016.

[18] S. Delcev and D. Draskovic, "Modern javascript frameworks: A survey study," in *2018 Zooming Innovation in Consumer Technologies Conference (ZINC)*, 2018, pp. 106–109.

[19] S. Ivanova and G. Georgiev, "Using modern web frameworks when developing an education application: a practical approach," in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2019, pp. 1485–1491.

[20] T.-D. Tripon, G. Adela Gabor, and E. Valentina Moisi, "Angular and svelte frameworks: a comparative analysis," in *2021 16th International Conference on Engineering of Modern Electric Systems (EMES)*, 2021, pp. 1–4.

[21] Google Developers, "Lighthouse," 2022, accessed 28 mar. 2022. [Online]. Available: https://developers.google.com/web/tools/lighthouse

[22] N. Nakajima, S. Matsumoto, and S. Kusumoto, "Jact: A playground tool for comparison of javascript frameworks," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, 2019, pp. 474–481.

[23] C. Q. Adamsen, A. Møller, S. Alimadadi, and F. Tip, "Practical ajax race detection for javascript web applications," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 38–48. [Online]. Available: https://doi.org/10.1145/3236024.3236038

[24] H. Brito, A. Gomes, Santos, and J. Bernardino, "Javascript in mobile applications: React native vs ionic vs nativescript vs native development," in *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, 2018, pp. 1–6.

[25] M. Corporation, "Mdn web docs glossary: Definitions of web-related terms," 2022, acessed in 21 march 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary

[26] D. Inc., “Docker engine release notes,” 2022, accessed in 21 march 2022. [Online]. Available: https://docs.docker.com/engine/release-notes/201012

[27] O. Foundation, “Releases,” 2022, accessed in 21 march 2022. [Online]. Available: https://nodejs.org/en/about/releases/

[28] F. Inc., “Releases,” 2022, accessed in 21 march 2022. [Online]. Available: https://docs.nginx.com/nginx/releases/

[29] SeleniumHQ, “Releases,” 2022, accessed in 21 march 2022. [Online]. Available: https://www.selenium.dev/downloads/

[30] I. Santos, J. C. C. Filho, and S. R. S. Souza, “A survey on the practices of mobile application testing,” in *2020 XLVI Latin American Computing Conference (CLEI)*, 2020, pp. 232–241.

[31] K. Sawant, R. Tiwari, S. Vyas, P. Sharma, A. Anand, and S. Soni, “Implementation of selenium automation amp; report generation using selenium web driver amp; atf,” in *2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, 2021, pp. 1–6.